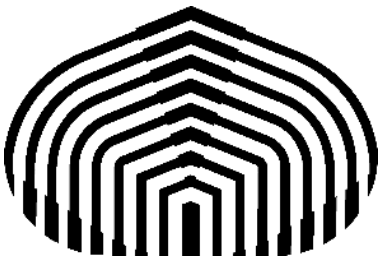

Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

Laboratorio de Algoritmos y Estructuras III CI-2693

Prof. Fernando Torre Mora

Especificaciones del Proyecto

Implementación del TAD Grafo Genérico Para Multígrafos

Integrantes:

José Barrera. Carnet: 15 - 10123

Jean Yazbek. Carnet: 15 – 11550

Sartenejas, octubre de 2018

Con la finalidad de implementar un TAD grafo genérico para multígrafos ajustado a las especificaciones del proyecto se ha estructurado el código en 12 archivos .java, a saber: Grafo, Arista, Arco, Lado, Vertice, GrafoDirigido, GrafoNoDirigido, Cliente, Transformer, TransformarBoolean, TransformarDouble, TransformarString. A continuación, se explicarán las decisiones de diseño y se indicarán los detalles más relevantes de la implementación realizada, con la finalidad de brindar al lector una idea general acerca de cómo funciona todo en conjunto.

En principio, se seguirá el orden en el que aparecen las implementaciones en el enunciado del proyecto:

- **Vertice<E>:** esta clase pública posee un constructor que juega el papel de la función solicitada Crear Vertice, y las funciones públicas getPeso, getId, getDato, toString, las cuales funcionan de manera trivial.
- **Lado<E>:** esta clase pública y abstracta posee las funciones públicas getPeso, getId, getDato, toString, las cuales no contienen ninguna instrucción.
- **Arco<E>:** esta clase pública extiende de la clase abstracta Lado, posee un constructor que juega el papel de la función solicitada Crear Arco, y las funciones públicas getExtremoInicial, getExtremoFinal, getPeso, getId, getDato, toString, las cuales funcionan de manera trivial.
- **Arista<E>:** análoga a la clase Arco<E>, se sustituye el nombre de las funciones getExtremoInicial, getExtremoFinal, por getExtremo1 y getExtremo2 para denotar que al tratarse de un grafo no dirigido en los lados no existe un orden.
- **Grafo<V,L>:** esta interfaz pública contiene todas las funciones indicadas, a saber: cargarGrafo, numeroDeVertices, numeroDeLados, agregarVertice, agregarVertice, obtenerVertice, estaVertice, estaLado, eliminarVertice, vertices, lados, grado, adyacentes, incidentes, clone, toString.
- **Nota1:** en la firma de la función cargarGrafo, se añadieron dos argumentos adicionales: transformer de tipo Transformer<String,V> y transformerarista de tipo Transformer<String,L>. La razón de implementar los transformadores se explica en la Nota2.
- **GrafoDirigido<V,L>:** esta clase pública es un implementación de la interfaz Grafo, y posee las siguientes operaciones particulares: agregarArco, agregarArco, eliminarArco,

obtenerArco, gradoInterior, gradoExterior, sucesores, predecesores. La idea general es que el digrafo es un diccionario (utilizamos el diccionario de Java: ***Hashtable***) donde las claves son objetos tipo vertice, y los valores son listas de objetos tipo lados. Es decir, cada vértice tiene asociado una lista de arcos donde dicho vértice es el extremo inicial.

- **GrafoNoDirigido<V,L>:** esta clase pública es una implementación de la interfaz Grafo, y posee las siguientes operaciones particulares: agregarArista, agregarArista, eliminarArista, obtenerArista. La idea general es análoga a la de dígrafo, pero cada Arista está presente en la lista de lados de ambos vertices, es decir toda arista tiene un duplicado, salvo los bucles. Es decir, cada vértice tiene asociado una lista de arcos donde dicho vértice es el extremo 1 o el extremo 2.
- **Cliente:** esta clase pública es la que le permite al usuario interactuar con los métodos del TAD. En ella se encuentran cuatro funciones: terminar, menuDirigido, menuNoDirigido y main.
- **Transformer:** esta interfaz pública, sirve para pasar datos de un tipo a otro.
- **TransformarBoolean:** esta clase pública es una implementación de la interfaz Transformer. Toma como argumento un String y lo transforma en un Boolean, a través de la función Boolean.valueOf().
- **TransformarDouble:** esta clase pública es una implementación de la interfaz Transformer. Toma como argumento un String y lo transforma en un Double, a través de la función Double.parseDouble().
- **TransformarString:** esta clase pública es una implementación de la interfaz Transformer. Toma como argumento un String y lo devuelve.
- **Nota2:** Los "transformadores" (la interfaz Transformer y las 3 clases que la implementan) son necesarios para el funcionamiento del programa ya que debido a uso de clases genéricas el compilador de Java arroja errores al tratar de utilizar directamente los métodos de conversión Boolean.valueOf() y Double.parseDouble() sobre las clases genéricas.

¿Cómo Ejecutar El Cliente?

El usuario luego de introducir: java Cliente, puede colocar seguidamente un nombre de archivo en caso de que desee cargar un grafo, o no colocarlo, en cuyo el programa asume que se desea crear un nuevo grafo. Luego si se desea crear grafo, se pedirá al usuario que indique el tipo

de dato para los vértices (pudiendo escoger entre Boolean, Double o String), luego el tipo de lado (nuevamente pudiendo escoger entre Boolean, Double o String), y finalmente el tipo de grafo (Dirigido o No). Una vez hecho esto, dependiendo del tipo de grafo seleccionado se mostrará o bien el menuDirigido o el menuNoDirigido ambos con la misma estructura, pero no con las mismas acciones disponibles, por obvias razones (las acciones sobre arcos en uno son sobre aristas en el otro, y menuNoDirigido carece de las opciones grado interno, grado externo, predecesores, sucesores). Después de seleccionar cada acción se volverá a desplegar el mismo menú. El usuario puede usar la opción salir para cerrar el programa.

Casos De Prueba:

Para probar que el programa se comporta como indica el enunciado. Se probaron las situaciones que se pide considerar obteniendo el resultado esperado. A saber:

- cargarGrafo: retorna true si los datos del archivo son cargados satisfactoriamente en el grafo, y false en caso contrario (problemas al abrir un archivo y el caso en el que el formato del archivo sea incorrecto).
- numeroDeVertices y numeroDeLados: si se aplica sobre un grafo vacío retornan 0.
- agregarVertice: si lo agrega al grafo retorna true, de lo contrario retorna false (el vértice ya existe).
- obtenerVertice: Se realiza una búsqueda lineal y retorna el vértice contenido en el grafo que posee el identificador id. Si no lo encuentra, se lanza la excepción NoSuchElementException(obtener un vertice que no esta en el grafo).
- estaVertice y estaLado: análogo al anterior, pero en lugar de retornar una excepción retorna false.
- eliminarVertice: análogo a estaVertice, pero se elimina una vez encontrado y todos los lados en los que se encontraba involucrado.
- vertices y lados: si se aplica sobre un grafo vacío retorna una lista vacía.
- grado, gradoInterior, gradoExterior, sucesores, predecesores, adyacentes e incidentes: análogos obtenerVertice en cuanto a la excepción.
- toString: si se aplica sobre un grafo vacío retornan un String vacío.
- agregarArco y agregarArista: si alguno de los vértices no existe, un lado con el mismo id ya existe o, vertices con el mismo id pero diferentes datos, retorna false. Si la inserción es exitosa retorna true.
- eliminarArco y eliminarArista: análogos a eliminarVertice.
- obtenerArco y obtenerArista: análogos a obtenerVertice.